Spring 2023

CS 151 Final Project: Picross Puzzle Game

Team: Amulya Pillutla (pillutla@stanford.edu), Jasmine Bilir (jbilir@stanford.edu), Ngoc Nguyen (ngocng@stanford.edu), and Si Yi Ma (siyima00@stanford.edu)

1. **Statement of the Problem Being Solved**
    a. Picross is a logic-based puzzle that challenges players to reveal a hidden picture by using number clues for each row and column. The objective is to fill in the correct cells based on the given clues. Replicating this puzzle digitally allows players to enjoy the game online and facilitates the development of additional features that make the game more accessible to all players of all skill levels.

2. **Why It Is Important**
    a. Replicating the Picross puzzle using logic programming holds educational value, particularly in the realms of mathematics and logic. The puzzle's reliance on numbers, deduction, and logical thinking can contribute to the development of essential skills and concepts. It provides an interactive environment where individuals can explore binary representation, logical operations, and problem-solving strategies, fostering learning in an enjoyable manner. The utilization of logic programming in the recreation of this puzzle serves as a showcase of the power and efficacy of this computational approach. It demonstrates how logic programming can be harnessed to create complex and engaging games, further highlighting its relevance and potential in various problem-solving domains.

3. **How Our Solution Addresses the Problem**
    a. The importance of our solution lies in providing a user-friendly, accessible platform for everyone including beginners and experts to play and solve the Picross puzzle. By implementing this logic programming solution, players can solve puzzles of the difficulty level of their choosing. The solution also introduces features that assist the user when they are stuck by either step by step revealing the puzzle or hinting as to which boxes are correct.

4. **Design of the System**
    a. **User Interface and Interaction**
        i. Create buttons for easy, medium, hard difficulties with their own colors
            1. Selection updates the lambda
        ii. Created a table, where we label each cell as mark(X,Y)or hint(X,Y)
            1. Section for number hints- hint(X,Y)
            2. Section for actual game- mark(X,Y)

iii. Clicking on a mark updates the lambda
    1. If the selected is correct, it replaces the correct(mark(X,Y)) with selected_cell(mark(X,Y)) in the lambda or it adds incorrect_cell(mark(X,Y))
iv. Correctly clicked cell turns blue, incorrectly selected cell turns red
v. Clicking the step button randomly reveals a correct cell, which is blue
vi. We have a checkbox option of hint or assist at the bottom, which updates the hover mode of the cursor to either grab or not-allowed depending on whether the mark being hovered is correct or incorrect
vii. Keep track of mistakes made above the grid, where if an incorrect cell is clicked, the number of mistakes is incremented by 1
viii. If the player selects all of the correct cells, then the correctly selected cells all turn blue and all incorrect cells turn white and a you win message pops up
ix. If the player selects the maximum number of incorrect cells allowed for that difficulty, then the whole grid turns black and a “You Lose :(“ message pops up

b. The Data

i. correct_cell(mark(X,Y))
ii. num_mistakes(0)

c. The Rules

i. Check if a cell is correct/incorrect when it is clicked
ii. Coloring the cell depending on correct/incorrect selection
iii. Output random correct and unselected cell when user selects step at bottom of page
iv. Show different cursor signifying correct/incorrect adjacent to any cell that has already been clicked when user selects hint at bottom of page
v. Show different cursor signifying correct/incorrect on any block when user selects assisted at bottom of page
vi. Make sure that user cannot select both hint and assist mode
vii. Counter for the number of mistakes the user makes in the picross
viii. Level selection between easy, medium, hard - hard cap on number of mistakes made my user
    1. Game doesn’t work if level is not selected
    2. Game turns black when user makes more than capped mistakes
ix. Display a “you win message” when user wins and a “you lose” message when user loses
x. Check if player has clicked all correct cells in order to display message

5. Challenges Faced in Building the Solution

a. We encountered many challenges throughout the development of this project. The first challenge we faced was getting the step feature and the hint features to function correctly. Since we must check every square and we wanted for there to be a degree of randomness and difficulty, figuring out how to logically progress (or reveal pieces of information) about the game state was tricky. Another challenge we faced that was a little bit more abstract was having to take one step or one move and extrapolating it to figure out the whole game setup. In another case, it would have been easier to, for example, have an array of information to manipulate.

6. **Relevance of Logic Programming to the Task**
    a. In terms of the Logic puzzle, Picross itself is a logic game that requires a player to think about the hints given and eliminate squares that might be blank. So, creating the most-simplistic game play in that sense already required a logic basis. For some bonus logic, we implemented features not native to the original picross game to make the logic programming more interesting. We were able to utilize concepts like recursive relationships etc. in order to implement different types of hints, autoplay steps, restrictions on different levels etc. And finally, for the DOM Interaction, because we wanted to incorporate many different types of interactions we had to investigate how to use logic programming to update the dom. We ultimately incorporated check-boxes, click interactions, live prompt updating, hover changes, and more

7. **How Logic Programming Technology Helped/Made the Work More Difficult?**
    a. Helped
        i. Picross is a logic game, so doing the logic programming for this project was not too difficult, as we were able to see how the logic came together
    b. Challenged
        i. Understanding the entire boardstate from a logic programming perspective was difficult. Coming from standard programming backgrounds, we all had to shift our perspectives from a world where board-state could be accessed as an array to thinking about relations and rules. In specific cases, this was especially difficult when considering hints and updating information where it may not have been in a different language. Additionally, logic programming proved to be a challenge in writing rules that were safe and did not produce bugs. Oftentimes we would discover bugs in the development process that would be niche or hard to track down because of the edge cases in our ruleset. This was just an additional way in which we need to reframe and become better at understanding exactly what our logical relations defined.